

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

(И.О. Фамилия)

Москва, 2024

Оглавление

Организация клиент-серверного взаимодействия между Golang и PostgreSQL.....	1
Цель работы.....	4
Ход работы.....	4
Задание.....	4
Микросервис hello:.....	4
Микросервис Query.....	9
Микросервис count.....	12
Вывод.....	16

Цель работы

Получение первичных навыков в организации долгосрочного хранения данных с использованием PostgreSQL и Golang.

Ход работы

Задание

Переделать коды из 6 лабораторной работы в микросервисы и организовать клиент-серверное взаимодействие между Golang и PostgreSQL.

Микросервис hello:

```
package main

import (
    "database/sql"
    "encoding/json"
    "flag"
    "fmt"
    "log"
    "net/http"

    _ "github.com/lib/pq"
)

const (
    host = "localhost"
    port = 5432
    user = "ps1"
    password = "1103"
    dbname = "sandbox"
)

type Handlers struct {
    dbProvider DatabaseProvider
}

type DatabaseProvider struct {
    db *sql.DB
}
```

```

// Обработчики HTTP-запросов
func (h *Handlers) GetHello(w http.ResponseWriter, r *http.Request) {
    msg, err := h.dbProvider.SelectHello()
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        w.Write([]byte(err.Error()))
    }

    w.WriteHeader(http.StatusOK)
    w.Write([]byte(msg))
}

func (h *Handlers) PostHello(w http.ResponseWriter, r *http.Request) {
    input := struct {
        Msg string `json:"msg"`
    }{}

    decoder := json.NewDecoder(r.Body)
    err := decoder.Decode(&input)
    if err != nil {
        if err != nil {
            w.WriteHeader(http.StatusBadRequest)
            w.Write([]byte(err.Error()))
        }
    }

    err = h.dbProvider.InsertHello(input.Msg)
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        w.Write([]byte(err.Error()))
    }

    w.WriteHeader(http.StatusCreated)
}

// Методы для работы с базой данных
func (dp *DatabaseProvider) SelectHello() (string, error) {
    var msg string

    // Получаем одно сообщение из таблицы hello, отсортированной в
    случайном порядке

```

```

row := dp.db.QueryRow("SELECT message FROM hello ORDER BY
RANDOM() LIMIT 1")
err := row.Scan(&msg)
if err != nil {
return "", err
}

return msg, nil
}
func (dp *DatabaseProvider) InsertHello(msg string) error {
_, err := dp.db.Exec("INSERT INTO hello (message) VALUES ($1)", msg)
if err != nil {
return err
}

return nil
}

func main() {
// Считываем аргументы командной строки
address := flag.String("address", "127.0.0.1:8081", "адрес для запуска
сервера")
flag.Parse()

// Формирование строки подключения для postgres
psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+
"password=%s dbname=%s sslmode=disable",
host, port, user, password, dbname)

// Создание соединения с сервером postgres
db, err := sql.Open("postgres", psqlInfo)
if err != nil {
log.Fatal(err)
}
defer db.Close()

// Создаем провайдер для БД с набором методов
dp := DatabaseProvider{db: db}
// Создаем экземпляр структуры с набором обработчиков
h := Handlers{dbProvider: dp}

```

```
// Регистрируем обработчики
http.HandleFunc("/get", h.GetHello)
http.HandleFunc("/post", h.PostHello)

// Запускаем веб-сервер на указанном адресе
err = http.ListenAndServe(*address, nil)
if err != nil {
log.Fatal(err)
}
}
```

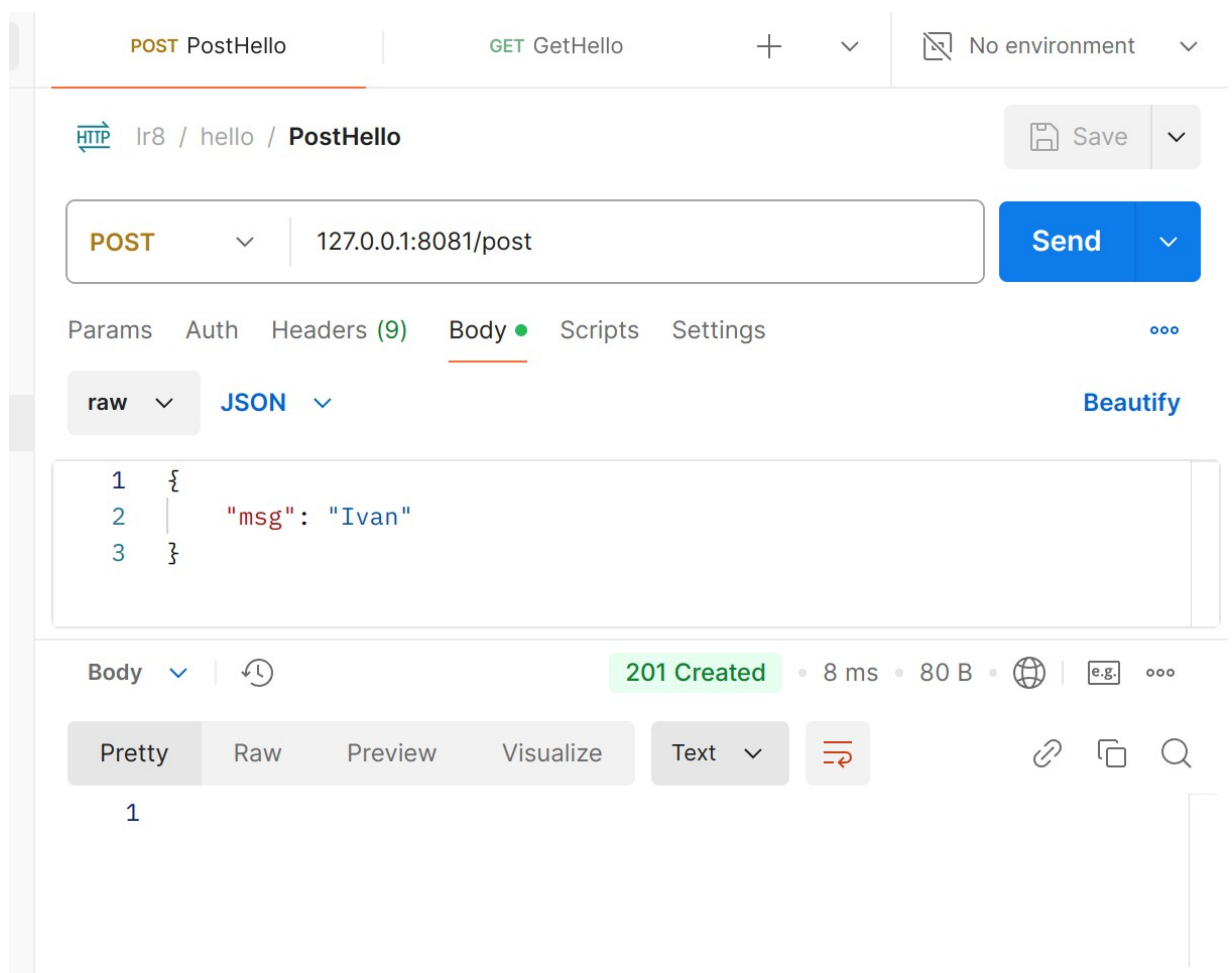


Рисунок 1 — результат Post-запроса hello

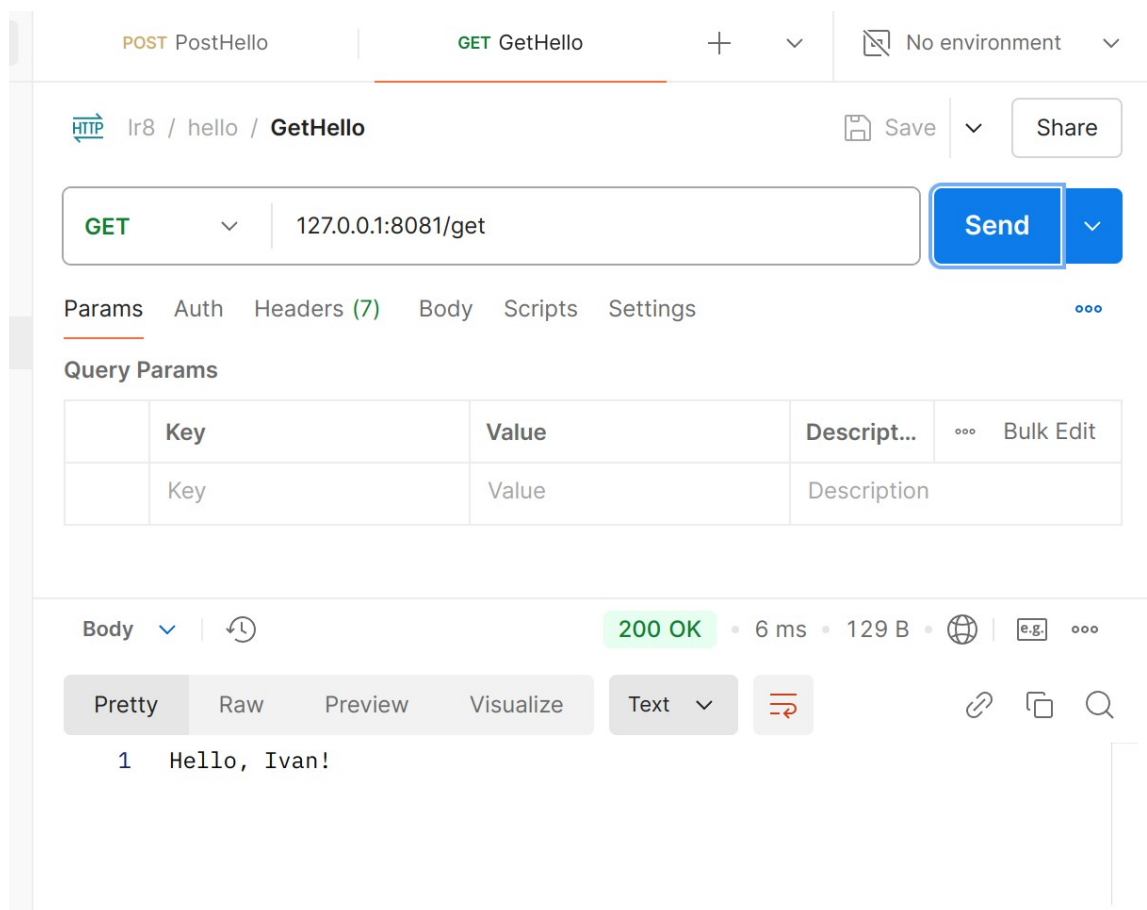


Рисунок 2 - результат выполнения Get-запроса hello

Микросервис Query

```
package main
```

```
import (  
    "database/sql"  
    "flag"  
    "fmt"  
    "log"  
    "net/http"
```

```
    _ "github.com/lib/pq"  
)
```

```
const (  
    host = "localhost"  
    port = 5432  
    user = "ps1"  
    password = "1103"  
    dbname = "lr8"  
)
```

```
type Handlers struct {  
    dbProvider DatabaseProvider  
}
```

```
type DatabaseProvider struct {  
    db *sql.DB  
}
```

```
func (h *Handlers) GetQuery(w http.ResponseWriter, r *http.Request) {  
    name := r.URL.Query().Get("name")
```

```
    if name == "" {  
        w.WriteHeader(http.StatusBadRequest)  
        w.Write([]byte("The parameter is not entered"))  
        return  
    }
```

```
    test, err := h.dbProvider.SelectQuery(name)  
    if !test {  
        w.WriteHeader(http.StatusBadRequest)  
        w.Write([]byte("The note has not been added to DB"))  
        return  
    }  
    if err != nil {
```

```
w.WriteHeader(http.StatusInternalServerError)
w.Write([]byte(err.Error()))
}
```

```
w.WriteHeader(http.StatusOK)
w.Write([]byte("Hello," + name + "!"))
}
```

```
func (h *Handlers) PostQuery(w http.ResponseWriter, r *http.Request) {
    name := r.URL.Query().Get("name")
    if name == "" {
        w.WriteHeader(http.StatusBadRequest)
        w.Write([]byte("The parameter is not entered"))
        return
    }
}
```

```
test, err := h.dbProvider.SelectQuery(name)
if test && err == nil {
    w.WriteHeader(http.StatusBadRequest)
    w.Write([]byte("The note has already been added to DB"))
    return
}
```

```
err = h.dbProvider.InsertQuery(name)
if err != nil {
    w.WriteHeader(http.StatusInternalServerError)
    w.Write([]byte(err.Error()))
}
```

```
w.WriteHeader(http.StatusCreated)
w.Write([]byte("Note added"))
}
```

```
func (dp *DatabaseProvider) SelectQuery(msg string) (bool, error) {
    var rec string
```

```
    row := dp.db.QueryRow("SELECT name_query FROM query WHERE name_query =
    ($1)", msg)
    err := row.Scan(&rec)
    if err != nil {
        return false, err
    }
}
```

```
    return true, nil
}
```

```

func (dp *DatabaseProvider) InsertQuery(msg string) error {
_, err := dp.db.Exec("INSERT INTO query (name_query) VALUES ($1)", msg)
if err != nil {
return err
}

return nil
}

func main() {
address := flag.String("address", "127.0.0.1:8083", "server startup address")
flag.Parse()

psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+
"password=%s dbname=%s sslmode=disable",
host, port, user, password, dbname)

db, err := sql.Open("postgres", psqlInfo)
if err != nil {
log.Fatal(err)
}
defer db.Close()
dp := DatabaseProvider{db: db}
h := Handlers{dbProvider: dp}

http.HandleFunc("/get", h.GetQuery)
http.HandleFunc("/post", h.PostQuery)

```

The screenshot shows a REST client interface with a top navigation bar containing tabs for different requests: POST PostHello, GET GetHello, GET GetQuery (selected), and POST PostQuery. A dropdown menu shows 'No environment'. Below the navigation bar, the URL bar shows 'Ir8 / query / GetQuery' and a 'Save' button. The main area shows a GET request to '127.0.0.1:8083/get?name=Ivan' with a 'Send' button. Below the request, there are tabs for Params, Auth, Headers (7), Body (selected), Scripts, and Settings. The Body tab shows a 'raw' dropdown and a 'JSON' dropdown. The response area shows a '200 OK' status, '3 ms' time, and '128 B' size. The response body is 'Hello, Ivan!'.

```
_ "github.com/lib/pq"  
)
```

```
const (  
host = "localhost"  
port = 5432  
user = "ps1"  
password = "1103"  
dbname = "lr8"  
)
```

```
type Handlers struct {  
dbProvider DatabaseProvider  
}
```

```
type DatabaseProvider struct {  
db *sql.DB  
}
```

```
func (h *Handlers) GetCount(w http.ResponseWriter, r *http.Request) {  
msg, err := h.dbProvider.SelectCount()  
if err != nil {  
w.WriteHeader(http.StatusInternalServerError)  
w.Write([]byte(err.Error()))  
}
```

```
w.WriteHeader(http.StatusOK)  
w.Write([]byte("counter: " + strconv.Itoa(msg)))  
}
```

```
func (h *Handlers) PostCount(w http.ResponseWriter, r *http.Request) {  
input := struct {  
Msg int `json:"msg"`  
}{}  

```

```
decoder := json.NewDecoder(r.Body)  
err := decoder.Decode(&input)  
if err != nil {  
w.WriteHeader(http.StatusBadRequest)  
w.Write([]byte(err.Error()))  
}
```

```
err = h.dbProvider.UpdateCount(input.Msg)  
if err != nil {  
w.WriteHeader(http.StatusInternalServerError)  
w.Write([]byte(err.Error()))  
}
```

```
w.WriteHeader(http.StatusCreated)  
w.Write([]byte("counter changed"))  
}
```

```

func (dp *DatabaseProvider) SelectCount() (int, error) {
var msg int

row := dp.db.QueryRow("SELECT number FROM counter WHERE id_number = 1")
err := row.Scan(&msg)
if err != nil {
return -1, err
}

return msg, nil
}

func (dp *DatabaseProvider) UpdateCount(msg int) error {
_, err := dp.db.Exec("UPDATE counter SET number = number + $1 WHERE id_number = 1", msg)
if err != nil {
return err
}

return nil
}

func main() {
address := flag.String("address", "127.0.0.1:8082", "адрес для запуска сервера")
flag.Parse()

psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+
"password=%s dbname=%s sslmode=disable",
host, port, user, password, dbname)

// Создание соединения с сервером postgres
db, err := sql.Open("postgres", psqlInfo)
if err != nil {
log.Fatal(err)
}
defer db.Close()

dp := DatabaseProvider{db: db}
h := Handlers{dbProvider: dp}

http.HandleFunc("/get", h.GetCount)
http.HandleFunc("/post", h.PostCount)

err = http.ListenAndServe(*address, nil)
if err != nil {
log.Fatal(err)
}
}

```

POST Post | GET GetHr | GET GetQ ● | POST Pos ● | GET GetC ● | POST Pos ● + ▾ No environment ▾

HTTP Ir8 / count / **PostCounter** Save ▾ Share

POST ▾ 127.0.0.1:8082/post Send ▾

Params Auth Headers (9) **Body ●** Scripts Settings Cookies

raw ▾ JSON ▾ Beautify

```
1 {
2   "msg": 30
3 }
```

Body ▾ ↺ 201 Created • 4 ms • 137 B • 🌐 📄 ⋮

Pretty Raw Preview Visualize Text ▾ ⌵

```
1 counter changed
```

Рисунок 5 - Post-запрос count

POST Post | GET GetHr | GET GetQ ● | POST Pos ● | **GET GetC ●** | POST Pos ● + ▾ No environment ▾

HTTP Ir8 / count / **GetCounter** Save ▾ Share

GET ▾ 127.0.0.1:8082/get Send ▾

Params Auth Headers (7) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	⋮ Bulk Edit
	Key	Value	Description	

Body ▾ ↺ 200 OK • 2 ms • 128 B • 🌐 📄 ⋮

Pretty Raw Preview Visualize Text ▾ ⌵

```
1 counter: 30
```

Рисунок 6 - Get-запрос count

Вывод

При выполнении заданий лабораторной были получены навыки по организации долгосрочного хранения данных с использованием PostgreSQL.